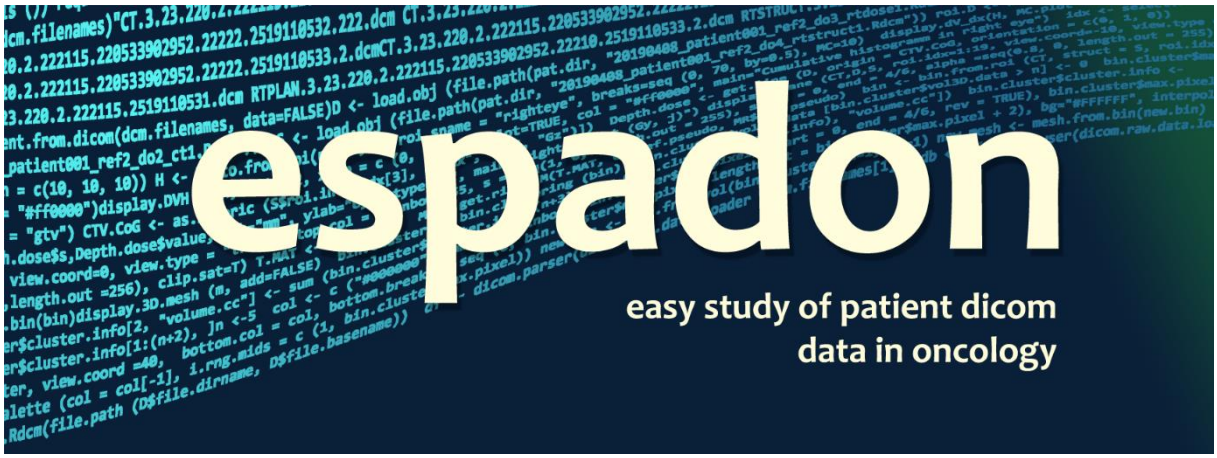


3D geometry

C. Fontbonne and J.-M. Fontbonne | LPC Caen
Normandie Univ, ENSICAEN, UNICAEN, CNRS/IN2P3, LPC Caen, 14000 Caen, France
Contact.espadon@lpccaen.in2p3.fr
https://espadon.cnrs.fr/

Table of content

1 Introduction. What we will learn?

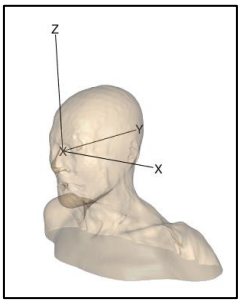
Before doing powerful computations on objects, the first job often consists in displaying data. For this, espadon incorporates lot of instructions allowing 2D and 3D representations of images and volumes. All of them are generally easy to use for simple jobs, but incorporate lot of options for tuning the result.

Despite its apparent simplicity, handling and representing 3D matrixes (as CT, rt-Dose…) can be challenging, especially when these objects were not acquired in the same Frame of Reference (FoR), which is the case for instance for MR versus CT and rt-dose.

Frames of reference (FoR) are fundamentals when patients have images coming from A) device and must be transferred to B) device for treatment or planning. Every things explained above are user-transparent, but it is important to understand the whole chain and its consequences on images transformations.

To understand what a good frame of reference is, for a patient lying on his back, looking up (for instance),

- X axis goes from right to left
- Y axis goes from face to back
- Z axis goes from feet to head
- Origin can be everywhere, usually, on the rotation axis/isocenter for rotating or helicoidal treatment machines



X, Y, Z form a **direct orthonormal reference frame**. “Direct” means $X \wedge Y = Z$; “ortho” is for $X.Y = X.Z = Y.Z = 0$; and “normal” as $|X| = |Y| = |Z| = 1$. **Coordinates units** are expressed in **millimeters** (in DICOM).

This is the case in DICOM-RT series. Unfortunately, images are always expressed as matrix points (raster coordinates system), not in space coordinates. That is where DICOM makes a useful bridge between images and space.

The first section will be dedicated to the formalism of images - how they are stored and used in espadon.

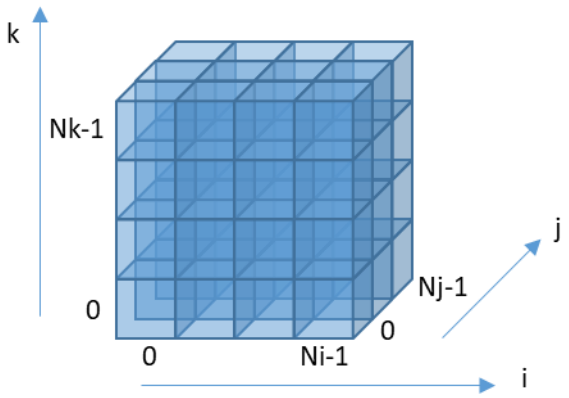
The second section will explore reference frame switching, which is particularly useful when using images from different machines simultaneously.

We will illustrate the possibilities offered by espadon, as well as their pitfalls.

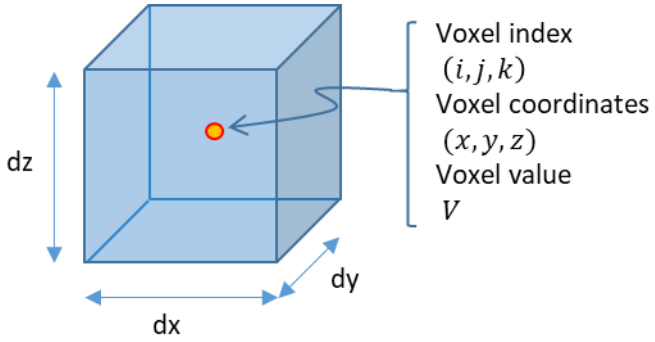
2 From image to space

2.1 How 3D series are exploited in DICOM

In DICOM format, i, j, k arrays begin at 0 and end at $(Ni-1, Nj-1, Nk-1)$ where Ni, Nj, Nk are the number of voxel in each direction:



The voxel value, V, is supposed to represent the measurement in the middle of the parallelepiped forming the walls of the voxel:



Finally, i, j, k can be converted into x, y, z space coordinates using the `xyz.from.ijk` matrix:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = {}^{xyz}M_{ijk} \cdot \begin{pmatrix} i \\ j \\ k \\ 1 \end{pmatrix}$$

Note: the `xyz.from.ijk` matrix is a 4x4 matrix able to make rotations, homotheties and translations. The fourths argument should always be 1.

In order to fill M_{ijk}^{xyz}, DICOM provides:

- The cosine directions of I and J axis, $\overrightarrow{C_I}$ and $\overrightarrow{C_J}$. The cosine directions of K axis is the vector product of the formers: $\overrightarrow{C_K} = \overrightarrow{C_I} \wedge \overrightarrow{C_J}$.
- The voxel size dx, dy, dz.
- And the position of the voxel $(i = 0, j = 0, k = 0)$: $\vec{O}$

This way, the matrix writes:

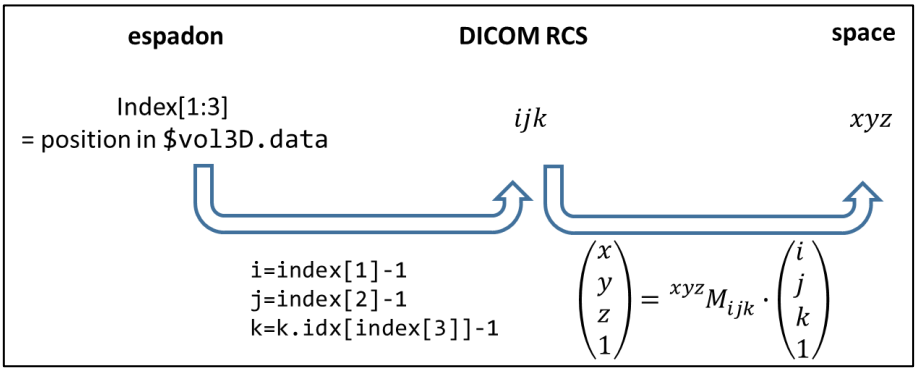
$${}^{xyz}M_{ijk} = \begin{pmatrix} dx \cdot c_i^x & dx \cdot c_j^x & dx \cdot c_k^x & O^x \\ dy \cdot c_i^y & dy \cdot c_j^y & dy \cdot c_k^y & O^y \\ dz \cdot c_i^z & dz \cdot c_j^z & dz \cdot c_k^z & O^z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2.2 How 3D series are stored in espadon

DICOM matrixes (CT, rt-Dose, MR, …) are stored in a unique three dimensions array in espadon (even CT or MR that are initially stored as planes). Arrays, in R, begin at index 1, last index being numbered as the number of elements in the current row. In the following, we will name "index" the position of a voxel in the array, as it is stored in R. Images are stored in the `$vol3D.data` field of `volume` class objects.

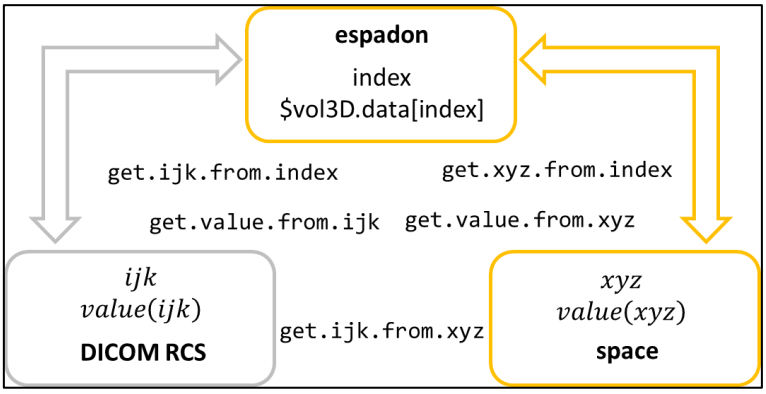
There is no direct link between R indices and DICOM ijk, as some planes can be missing or k coordinate can go in decreasing order (depending on the TPS). If you need to access the content of the images, **it is recommended that you use the mechanisms offered by espadon** which we will describe. For the moment, remember that we have three coordinate systems:

- space coordinates, xyz, linked to the patient frame of reference
- Raster Coordinates System, ijk, linked to DICOM indices
- Espadon coordinates, $index$, linked to the array `$vol3D.data` as stored in espadon.



Coordinates systems in espadon from a mathematical point of view.

All the instructions you should need are sum up on the following relation map:



Instructions that can be used to change coordinates system, or to retrieve image value in a given coordinates system. The main pathway links espadon to space (in orange). The passage through the RCS (in grey) is mainly used internally and is generally of little interest to the user.

Let us illustrate the main functionalities on a MR, whose transform matrix and k translation vector are:

```
MR$xyz.from.ijk

         [,1]        [,2]        [,3]      [,4]
x0 0.01942183  0.02102505 -0.99841032   73.2606
y0 0.50255341  0.06936294  0.04357673 -169.6920
z0 0.07016882 -0.50260064 -0.03575208   51.6103
   0.00000000  0.00000000  0.00000000    1.0000

MR$k.idx

  [1]   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
 [19]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35d
...
```

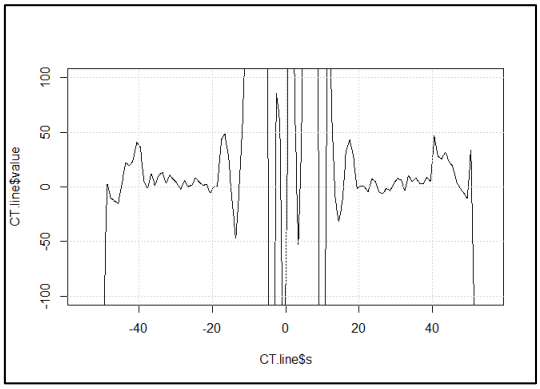
The relation between espadon indices, space coordinates and image value are immediate. The following example retrieves the maximum value of a stack of images, whether in espadon or spatial coordinates:

```
idx <- which.max(MR$vol3D.data)
idx

[1] 20752204

xyz <- get.xyz.from.index(idx, MR)
xyz

            x        y        z
[1,] 2.559889 5.852864 30.29591

MR$vol3D.data[idx]

[1] 5351

get.value.from.xyz(xyz, MR)

[1] 5351
```

So, if you are working in spatial coordinates, you always have a way to get the data from the images and vice versa. We have illustrated this mechanism on a point (the one containing the maximum value of the array), but espadon has two other useful instructions for doing this work along a line or on the surface of a plane.

Let us illustrate the usage of `get.line`, for retrieving data on a line. In this example, we want to draw the CT on a line from the center of gravity of the left eye to the center of gravity of the right eye. We also have the rt-struct loaded in `S`:

```
G.right <- S$roi.info[S$roi.info$roi.pseudo == "oeild", c("Gx", "Gy", "Gz")]
G.right <- as.numeric (G.right) # as.numeric mandatory as G is a data.frame, not a vector
G.left <- S$roi.info[S$roi.info$roi.pseudo == "oeilg", c("Gx", "Gy", "Gz")]
G.left <- as.numeric (G.left)
origin <- (G.right + G.left) / 2
u <- G.right - G.left
CT.line <- get.line (CT, origin=origin, orientation = u, grid = seq (-55, 55, by=.1))

dev.new (width = 7, height = 5, noRStudioGD = TRUE)
par (mar=c(4, 4, 1, 1))
plot (CT.line$s, CT.line$value, typ="l", ylim=c(-100, 100))
```

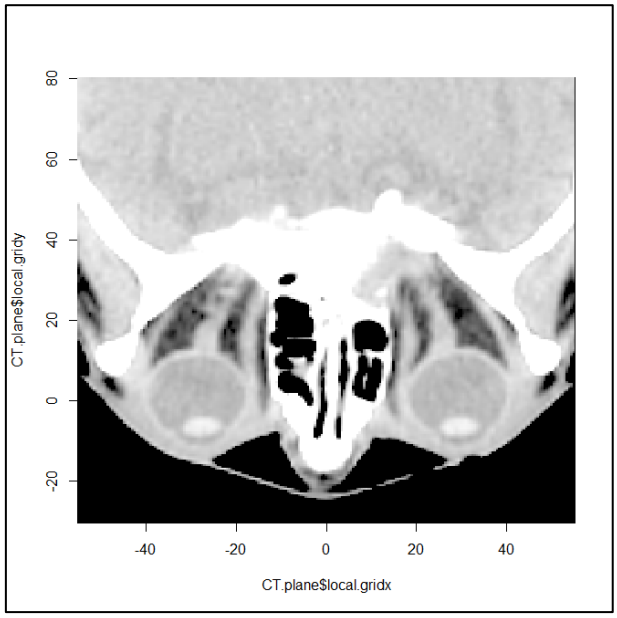


CT value along a line from left eye to right eye. Abscissa units are in mm. Left eye is the area near 0 HU in the range s=[-40, -20] mm. Right eye is at s=[20, 40]mm.

The `get.line` instruction needs the origin point (just at the middle of both eyes), the orientation vector (from left to right as `u <- G.right - G.left`) and the grid, expressed in curvilinear abscissa, where you want the value. Note the orientation vector is internally normalized. Then you get a list with as many points you requested, their spatial coordinates, their curvilinear abscissa, expressed in mm from the origin, along the requested direction, and the value (here of the CT). It is up to you to exploit the information, for instance, if you want, for any reason, to measure the eye radius along this line.

If you need to retrieve the data along any plane, we suggest you use the `get.plane` instruction in conjunction with `orientation.create`. The function `orientation.create` helps forming the base vectors needed for the plane. It just needs 3 points in space. The first one will be left eye, the second one, right eye, and the last one, the chiasma center of gravity. Applying this function, you will get x coordinates vector joining the first point to the second one, and y coordinates vector the orthonormal vector in the direction from first point to third point:

```
# G.left & G.right were defined above
G.chias <- S$roi.info[S$roi.info$roi.pseudo == "chiasma", c("Gx", "Gy", "Gz")]
G.chias <- as.numeric (G.chias)
uv <- orientation.create (A = G.left, B = G.right, C = G.chias)
CT.plane <- get.plane (CT, origin = origin, uv,
                       xgrid=seq (-55, 55, by=0.5), ygrid=seq (-30, 80, by=0.5))
dev.new(width = 7, height = 7, noRStudioGD = TRUE)
par (mal=c(4, 4, 1, 1))
image (x=CT.plane$local.gridx, y=CT.plane$local.gridy,
       z=CT.plane$vol3D.data[,,1],
       col=grey.colors(255, start=0, end=1), breaks = c(-1e6, seq (-100, 100, length.out = 254), +1e6))
```



CT value of the plane joining left eye (at x=-30, y=0) to right eye (at x=+30, y=0) and containing the chiasma (not visible on the CT).

Note that, in the call to `orientation.create`, we decided to put left eye at first position and right eye in the second one. That is why left eye is at x<0 and right eye at x>0. They are both centered on the line y=0.

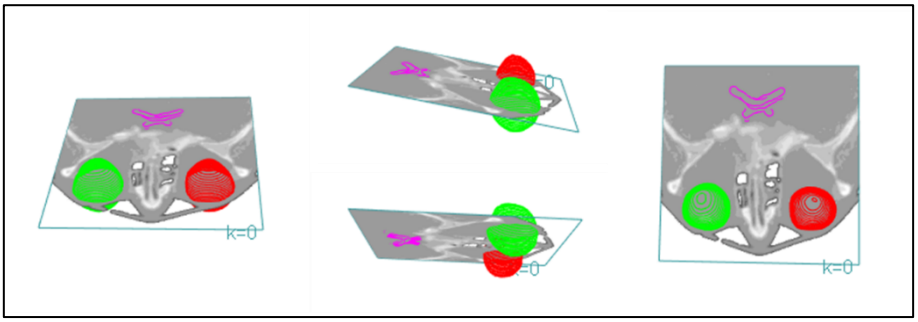
When calling `get.plane`, we get `$local.gridx` and `$local.gridx` which are the local coordinates calculated along local reference frame whose origin was in the middle of both eyes. Both can be used to label coordinates when making images. Note that the cut plane is stored in `$vol3D.data` which is an array. To get it as a matrix instead (for imaging, for instance), just use `$vol3D.data[,,1]`.

To display the image, you could have also used the display.kplane function as below :

```
display.kplane (CT.plane, pt00 = c (CT.plane$local.gridx[1], CT.plane$local.gridy[1]),
                dxy = CT.plane$dxyz[1:2], abs.lab = "x", ord.lab = "y",
                breaks = seq (-100, 100, length.out = 256), sat.transp = TRUE,
                main ="my plane")
```

When playing with cuts (whatever lines or planes), it is very easy to make reference frame flips or mirrors. It is then often useful to check, directly in 3D, that the result you obtain actually corresponds to your expectations in terms of right/left, up/down on the resulting image. To do this, do not hesitate to make a quick inspection, for example:

```
display.3D.stack (CT.plane)
S$roi.info[S$roi.info$roi.pseudo == "oeilg", "color"] <- "red"          # left eye = red
S$roi.info[S$roi.info$roi.pseudo == "oeild", "color"] <- "green"        # right eye = green
S$roi.info[S$roi.info$roi.pseudo == "chiasma", "color"] <- "magenta"    # chiasma = magenta
display.3D.contour(S, roi.name = c("oeilg", "oeild", "chiasma"))
```



3D representation of the cut plane in conjunction to right eye (we colored in green), left eye (we colored in red) and chiasma (in magenta). The nasal cavities are clearly visible and allow us to check the orientation of the image obtained above (right/left, in particular).

By carrying out these checks, we are reasonably confident in the calculation of the plane we just made. We can therefore exploit its information using the image and local coordinates provided by espadon.

3 From space to space; change of reference frame

Patient examinations (MR, CT...) are generally performed on machines that are not linked to each other. Fortunately, in the course of his or her work, the radiation oncologist usually merges the images from the different machines. This fusion leads to the creation of a transformation matrix that can be saved in a reg file in DICOM format and used by espadon to collate the reference frames.

At present, espadon only supports rigid transformations involving rotations and translations, which is usually the case when merging MR with a processing CT, for example.

3.1 the algebra of rigid transformations

Suppose we have images of a patient obtained on machine A that we wish to merge with those from machine B. This operation requires the transformation of spatial coordinates from A to B. This transformation takes place through a transfer matrix from A to B which is a 4x4 matrix stored in the reg file.

Mathematically speaking:

$$\begin{pmatrix} x_B \\ y_B \\ z_B \\ 1 \end{pmatrix} = {}^{B}M_{A} \cdot \begin{pmatrix} x_A \\ y_A \\ z_A \\ 1 \end{pmatrix}$$

The inverse operation (transformation from B to A) is simply:

$$\begin{pmatrix} x_A \\ y_A \\ z_A \\ 1 \end{pmatrix} = \left({}^{B}M_{A}\right)^{-1} \cdot \begin{pmatrix} x_B \\ y_B \\ z_B \\ 1 \end{pmatrix}$$

In addition, transformations can also be chained. If, for example, we have the matrices from A to B and from B to C, then:

$$\begin{pmatrix} x_C \\ y_C \\ z_C \\ 1 \end{pmatrix} = {}^{C}M_{B} \cdot {}^{B}M_{A} \cdot \begin{pmatrix} x_A \\ y_A \\ z_A \\ 1 \end{pmatrix}$$

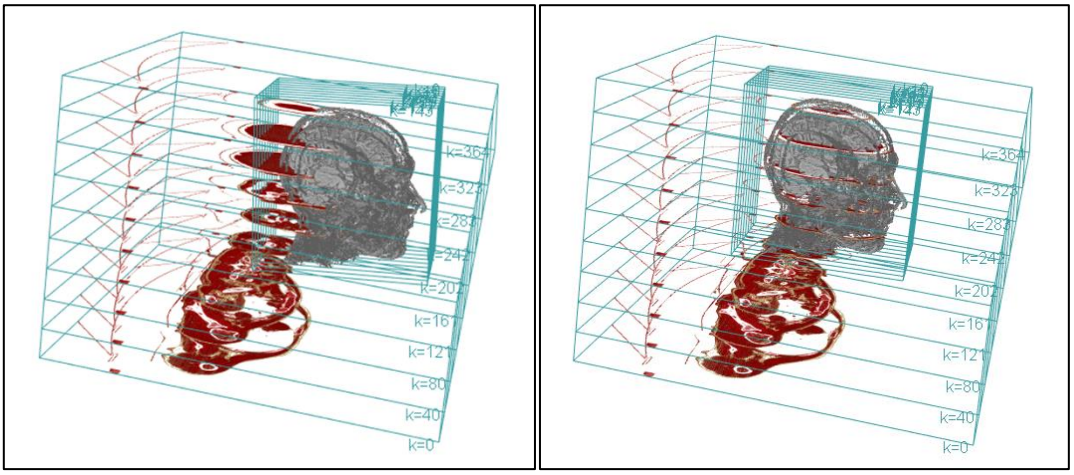
Despite this simplicity, handling reference frame changes can quickly become a headache if you have to remember the sometimes complex path from one reference frame to another. Espadon therefore has relatively intuitive features to limit this complexity. It does all the work for you.

3.2 How reference frame changes are handled in espadon

When you load the patient-oriented view, espadon analyses the reference frames used by the objects. All objects in the same reference frame can be used in parallel.

In addition, espadon also searches for reg files. These will connect the reference frames to each other through the `$T.MAT` field which will contain all the possible transformations. If objects are not in the same reference frame, but there is a path from one to the other, then you can also use them in parallel. All you have to worry about is choosing the reference frame you want to work in. Usually this is the treatment frame of reference. To illustrate this, let's run the code below:

```
require (espadon)
require (rgl)
pat.dir <- choose.dir ()
pat <- load.patient.from.Rdcm (pat.dir)

CT <- load.obj.data(pat$ct[[1]])
MR <- load.obj.data(pat$mr[[1]])

## MR and CT don't share the same reference frame (image on the left)
open3d (windowRect = c(1, 1, 600, 600))
display.3D.stack (CT, col= pal.RVV (200, alpha = c (rep(0,90), rep (1, 110))),
                  breaks = seq (-1000, 1000, length.out = 201))
display.3D.stack (MR)
rgl.snapshot ("dummy1.png")
old.par <- par3d (no.readonly = TRUE)

## now, they are correctly displayed (image on the right)
open3d (windowRect=c(1, 1, 600, 600))
display.3D.stack(CT, col= pal.RVV (200, alpha = c(rep(0,90), rep (1, 110))),
                 breaks = seq (-1000, 1000, length.out = 201))
display.3D.stack (MR, T.MAT = pat$T.MAT, display.ref = CT$ref.pseudo)
par3d (old.par)
rgl.snapshot("dummy2.png")
```



A CT (in false colours using the RVV palette) and a MR (in grey). On the left, the images are not superimposed because the acquisition reference frames are different. On the right, the use of the T.MAT mechanism allows the images to be merged as they were used by the radiotherapist oncologist.

Most espadon instructions include the T.MAT mechanism. When it is not set (by default), most instructions just check that the objects handled are in the same reference frame (e.g. an rt-dose and a treatment CT). They will return an error if this is not the case. When you pass them an object and the calculations must be performed in a defined reference frame, you must pass `$T.MAT`, and define the target reference frame by filling in the `display.ref` option.

3.3 Changing reference frame vs regriddind & other non-linear operations

In the previous paragraph, we discussed the changes in frames of reference, focusing on the how, but without going into detail on the why. The first answer to why that might come to mind is to correctly superimpose images that should be superimposed, and this is precisely the role of the illustration given above. However, we are addressing here programmers who will want to use the images to go further, e.g. execute selections, retrieve their contents, merge information, etc. We will see that all of this is possible with espadon, and that there are generally several ways to do this, some better than others.

Firstly, it must be understood that changes of reference frames are purely geometrical operations on coordinate systems. They do not affect the content of the images. They can be performed on the images as well as on the contours of the rt-struct, without loss of information.

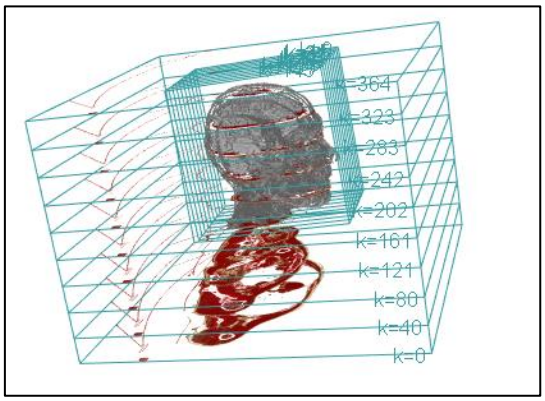
On the other hand, binary selections and resampling are non-linear operations on images. They are always necessary for data processing, but they inevitably lead to image modification and information loss or transformation. Even the so-called “trilinear” interpolation is not linear as it is not usually reversible!

Thus, when you work on objects located in different reference frames, it is in your interest to pay close attention to the order of the operations you are going to perform, so as to preserve the initial information as much as possible. We will go into more details in the following paragraphs.

3.3.1 Exporting an object in a given reference frame

Changing the reference frame of an image is an immediate operation as it only involves the orientation of the patient and its origin. The images themselves are not modified. This transformation uses the `vol.in.new.ref` function which creates an object of class `volume` in the new reference frame. In the following example, we will transport an MR in the CT reference frame. From then on, calculations performed in the CT frame of reference can also be performed on the MR, such as the analysis along a line or a plane, since the coordinate systems are now identical.

```
MR.in.CT <- vol.in.new.ref (MR, CT$ref.pseudo, pat$T.MAT)
display.3D.stack (CT, col= pal.RVV (200, alpha = c (rep(0,90), rep (1, 110))),
                  breaks = seq (-1000, 1000, length.out = 201))
display.3D.stack (MR.in.CT)
```

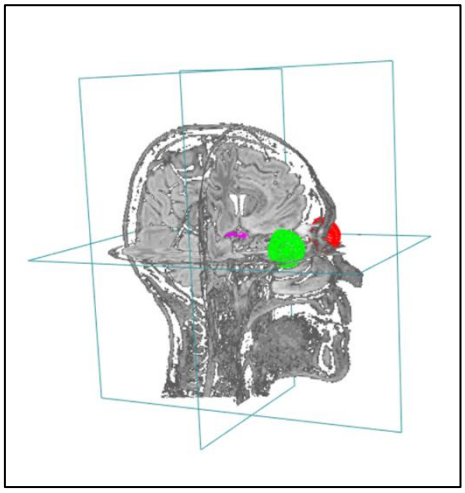


3D representation of the CT and the MR, in the CT frame of reference.

Although the images have not been modified (note here that the MR slices are still in sagittal view), the `MR.in.CT` imaging is indeed in the CT frame of reference.

In the same way, it would have been possible to transport the structures from the CT to the MR reference frame, using `struct.in.new.ref`, for example. This operation is a bit more time consuming because `min.x`, `max.x`, `min.y`, `max.y`, `min.z`, `max.z`, `Gx` , `Gy`, `Gz` are recalculated for all RoIs in the new reference frame.

```
S.in.MR <- struct.in.new.ref (S, MR$ref.pseudo, pat$T.MAT)
display.3D.sections(MR)
S.in.MR$roi.info[S.in.MR$roi.info$roi.pseudo == "oeilg", "color"] <- "red"
S.in.MR$roi.info[S.in.MR$roi.info$roi.pseudo == "oeild", "color"] <- "green"
S.in.MR$roi.info[S.in.MR$roi.info$roi.pseudo == "chiasma", "color"] <- "magenta"
display.3D.contour (S.in.MR, roi.name = c("oeild", "oeilg", "chiasma"))
```



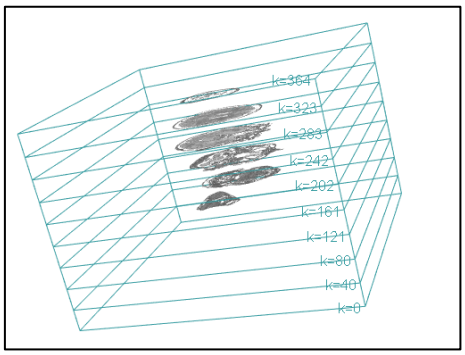
Frontal, sagittal and transverse view at x=0, y=0, z=0. Contours of eyes and chiasma

This "new" structure can of course be used to make selections on the MR.

3.3.2 Regridding & resampling

Another way of representing a volume in a reference frame is to sample it on that reference frame. This is done using the `vol.regrid` instruction. It allows the resampling of an image sequence on another:

```
MR.in.CT <- vol.regrid (MR, CT, pat$T.MAT)
display.3D.stack (MR.in.CT)
```



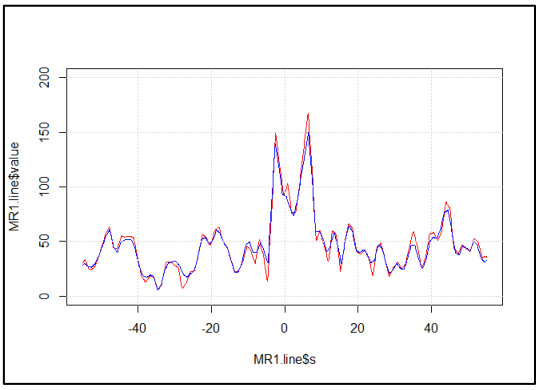
Resampling of the MR on the CT grid.

Note this time, the MR planes are aligned with CT planes. Both share the same space sampling.

Now, it is time to compare both strategies. We will plot the content of the images along the line from the left eye to the right eye, on the one hand for the MR that we have transported to the CT frame of reference, and on the other hand on the MR that we have resampled on the CT.

```
G.right <- S$roi.info[S$roi.info$roi.pseudo == "oeild", c("Gx", "Gy", "Gz")]
G.right <- as.numeric(G.right)
G.left <- S$roi.info[S$roi.info$roi.pseudo == "oeilg", c("Gx", "Gy", "Gz")]
G.left <- as.numeric (G.left)
origin <- (G.right + G.left) / 2
u <- G.right - G.left

MR.in.CT1 <- vol.in.new.ref (MR, CT$ref.pseudo, pat$T.MAT)
```

```
MR.in.CT2 <- vol.regrid (MR, CT, pat$T.MAT)

MR1.line <- get.line (MR.in.CT1, origin=origin, orientation = u, grid = seq (-55, 55, by=.1))
MR2.line <- get.line (MR.in.CT2, origin=origin, orientation = u, grid = seq (-55, 55, by=.1))

plot (MR1.line$s, MR1.line$value, typ="l", ylim=c(0, 200), col="red")
lines (MR2.line$s, MR2.line$value, typ="l", col="blue")
grid ()
```



MR along the "eyes line". In red, we have simply changed the reference frame of the MR. In blue, we resampled it to the CT grid. This last operation modified the content of the image.

This example illustrates perfectly the transformations brought about by the resampling (in blue) compared to the original content (in red). Of course, we always end up resampling the object, if only to obtain its value along the measurement line. However, it is now clear that it is better to do this as late as possible and as few times as possible to keep the information as intact as possible.

Ultimately, placing two objects on the same spatial grid is only useful for voxel-by-voxel comparison, which can be useful for labelling tissues (MR + CT), for example, or for modelling the effect of a 3D dose distribution on tissues (rt-dose + MR or CT)

4 What we learned

Objects sharing the same Frame of Reference (the `$ref.pseudo` field) are easy to use simultaneously in espadon. Depending on your needs you can get values at given coordinates in space, on a point, a line, a plane (`get.value.from.xyz`, `get.line`, `get.plane`), using various interpolation mechanisms.

When objects do not share their FoR, you will need to transport one object in the FoR of the other. This supposes the existence of a registration matrix, usually defined in a DICOM-reg file. If it exists, this matrix can be loaded using `load.T.MAT` instruction (if your data were converted into Rdcm format) or directly retrieved from the patient-oriented view, `pat$T.MAT`. Then most functions incorporate an automated mechanism for simultaneous usage of two objects (for instance `display.plane (…, T.MAT=The.T.MAT.I.loaded, display.ref="The.FoR.I.want", …)`). And last, it is always possible to transport an object into a given FoR in order to create a new object, using for instance `vol.in.new.ref`.

Do not hesitate to have a look at the reference manual in order to learn about options and maybe discover useful instructions not developed in this text…

You can see also:

```
get.ijk.from.index      Conversion of the indices of a point into ijk vector
get.ijk.from.xyz        Indices relating to the coordinates of the points
get.line                Image value along an axis
get.plane               Extracting a plane from a volume
get.rigid.M             Transfer matrix between two frames of reference
get.value.from.ijk      Value of the volume at a selection of DICOM indices
get.value.from.xyz      Voxel values on a selection of points
get.xyz.from.index      Conversion of the indices of a point, into xyz coordinate vector in the patient's
frame of reference
load.T.MAT              Loading of information about transfer matrices between frames of reference of patient
objects.
orientation.create      Creation of orientation The 'orientation.create' function creates the orientation
vector of a plane, from 3 points.
ref.add                 Adding a frame of reference in T.MAT
ref.cutplane.add        Adding volume's cutting planes frame of reference in T.MAT
ref.remove              Deletion of a frame of reference in T.MAT
ref.srctodest.add       Linking two existing frames of reference in T.MAT
save.T.MAT              Save a T.MAT class object
struct.in.new.ref       Change of frame of reference of a 'struct' class object.
vol.in.new.ref          Change of frame of reference of a volume
vol.regrid              Transform the grid of a volume class object into the grid of another
```